RansomClave: Ransomware Key Management using SGX

Alpesh Bhudia
Information Security Group
Royal Holloway, University of London
alpesh.bhudia.2018@live.rhul.ac.uk

Daniel O'Keeffe
Royal Holloway, University of London
daniel.okeeffe@rhul.ac.uk

Daniele Sgandurra
Information Security Group
Royal Holloway, University of London
daniele.sgandurra@rhul.ac.uk

Darren Hurley-Smith
Information Security Group
Royal Holloway, University of London
darren.hurley-smith@rhul.ac.uk

ABSTRACT

Modern ransomware often generate and manage cryptographic keys on the victim's machine, giving defenders an opportunity to capture exposed keys and recover encrypted data without paying the ransom. However, recent work has raised the possibility of future *enclave-enhanced* malware that could avoid such mitigations using emerging support for hardware-enforced secure enclaves in commodity CPUs. Nonetheless, the practicality of such enclave-enhanced malware and its potential impact on all phases of the ransomware lifecyle remain unclear. Given the demonstrated capacity of ransomware authors to innovate in order to better extort their victims (e.g. through the adoption of untraceable virtual currencies and anonymity networks), it is important to better understand the risks involved and identify potential mitigations.

As a basis for comprehensive security and performance analysis of enclave-enhanced ransomware, we present RansomClave, a family of ransomware that securely manage their cryptographic keys using an enclave. We use RansomClave to explore the implications of enclave-enhanced ransomware for the key generation, encryption and key release phases of the ransomware lifecycle, and to identify potential limitations and mitigations.

We propose two plausible victim models and analyse, from an attacker's perspective, how RansomClave can protect cryptographic keys from each type of victim. We find that some existing mitigations are likely to be effective during the key generation and encryption phases, but that RansomClave enables new *trustless* key release schemes that could potentially improve attacker's profitability and, by extension, make enclaves an attractive target for future attackers.

1 INTRODUCTION

Over the last decade, ransomware has become one of the more prominent cyber-security threats against individuals and organisations [28]. Ransomware is a malware that either renders a victim's computer resources (*locker-ransomware*) or valuable data (*crypto-ransomware*) unusable [36]. Crypto-ransomware is currently the most common of the two, and is especially difficult to defend against since victims are required to obtain the corresponding decryption key to access their data (assuming no backup is available).

A typical ransomware attack lifecycle goes through four main phases: installation, unique public/private key generation, encryption (with symmetric keys), and extortion/private key release. For a successful operation, the private key created in the second phase needs to be securely stored and only released in the last phase once the victim has paid the ransom [1, 33]. Previous work has shown several weaknesses in existing ransomware cryptographic key management systems [6, 8, 19, 23]. For example, many ransomware variants (e.g., DMA Locker, Locky, Cerber, WannaCry, NotPetya, and BadRabbit) generate and store cryptographic keys (both asymmetric and symmetric ones) in untrusted memory on the victim's machine before permanently deleting them. Fortunately, victims can exploit such weaknesses to extract the generated keys using memory forensics techniques [37] and tools like Dumpit, RAMCapturer and FTK imager [57]. Additionally, researchers have developed techniques, e.g. PayBreak [31], to capture keys generated at run-time.

Unfortunately, emerging technology could render some of these mitigations ineffective. As recent work has observed [38], ransomware authors share many of the challenges faced by *confidential cloud computing* [49], which seeks to protect applications deployed on remote cloud infrastructure from a malicious cloud provider. To solve these challenges, confidential cloud computing leverages emerging support on commodity hardware for trusted execution environments (TEEs), such as Intel SGX [16, 41]. SGX TEEs, commonly known as *enclaves*, allow parts of an application to be executed securely irrespective of the rest of the system [49]. As has been shown, it is also possible for malware to hide malicious activities within enclaves to evade anti-virus systems [38]. However, previous studies have not considered the impact of enclaves on the complete lifecycle of ransomware cryptographic key management, and the practicality and performance overhead of such a ransomware variant is unclear. Given the damage ransomware causes, it is important to understand what new advantages such ransomware might provide to attackers, whether anti-malware mechanisms in existing enclave technology are robust against them, and whether additional anti-ransomware mitigations might be required.

We perform an in-depth study of the potential implications of enclave-enabled ransomware using RansomClave, a proof-of-concept family of ransomware variants that leverages SGX to manage ransomware cryptographic keys on a victim's machine. RansomClave, once executed, creates an enclave that it uses to securely generate asymmetric key pairs. RansomClave releases the private key only when a ransom payment transaction from the Bitcoin blockchain is verified by the enclave.

In contrast to previous work, we analyse the security and practicality of RansomClave with respect to all phases of the ransomware lifecycle. During the key generation and encryption phases,

we show that there is a design-space whereby to avoid triggering network monitoring software a ransomware author may skip SGX remote attestation and avoid command and control (C2) server communication before encryption completes, but that this prevents dynamic loading of encrypted ransomware code to the enclave. Similarly, during the key release phase, we use RansomClave to show that enclaves open up a design-space of new key release schemes, each with different trade-offs for the attacker between security, reliability, and operational overhead. Of particular concern are fully autonomous or *trustless* key release schemes, which do not require the victim to trust the attacker to release the decryption key after a ransom is paid. This guarantee could increase a victim's willingness to pay a ransom and by extension attacker profitability.

Contributions. In summary, this paper makes the following contributions:

(1) We analyse key management schemes of existing ransomware variants and, based on our findings, derive requirements for a successful attack (§3).
(2) We introduce RansomClave, a proof-of-concept SGX-based ransomware that generates keys in an enclave to protect them from disclosure (§5), and analyse its performance.
(3) We propose three blockchain-based RansomClave key release schemes and show how avoiding interaction with the attacker post-infection could potentially make the ransomware operation more profitable for attackers (§6).
(4) We analyse mitigations to defend against SGX-based ransomware attacks (§8).

2 BACKGROUND

This section provides background on ransomware encryption mechanisms, and a summary of relevant features of Intel SGX.

Ransomware Encryption Schemes. Modern ransomware are often classified into three types based on the type of encryption employed. *Symmetric encryption ransomware* uses one key for both encryption and decryption and allows faster file encryption, resulting in shorter time to complete the attack. This also reduces the chances of the attack being discovered by requiring less processing power for computations. *Asymmetric encryption ransomware* uses asymmetric cryptography, with the public key used to encrypt files and the associated private master key to decrypt them. The encryption process is a resource-intensive task and slower in comparison to symmetric key encryption due to the overhead of public-key encryption. However, it is more secure since the encryption process can be completed using only the public key thus allowing the private key to be kept secret. Finally, *hybrid encryption ransomware* incorporates both symmetric encryption and asymmetric encryption. It uses symmetric encryption to encrypt the user files quickly. The symmetric key is then encrypted using asymmetric encryption. Hybrid encryption is often used by newer strains of ransomware, and reaps the benefits of both symmetric encryption (speed) and asymmetric encryption (security) [9, 30].

Intel SGX. Intel SGX is a security enhancement to recent Intel CPUs that provides a trusted execution environment (TEE). SGX TEEs, known as *enclaves*, protect the integrity and confidentiality of applications handling sensitive data such as cryptographic keys. The SGX architecture enables an application to instantiate one or more enclaves such that enclave code and data is protected from the OS and hypervisor [16, 54], and even from an attacker with physical access to the victim's computer [39].

Enclave code must execute in user space, and is not able to execute system calls or access secure peripherals. As such, applications are divided into two parts, a protected enclave (trusted code) and an unprotected part (untrusted code) which handles communication between the OS and enclave. The processor transparently encrypts and integrity protects enclave data whenever it leaves the CPU. The Intel SGX SDK provides a function call-like abstraction for entering and exiting an enclave. Calls into the enclave are referred to as `ECALL`s (enclave entry call), and calls from the enclave to outside as `OCALL`s (outside call) [15, 39, 52].

The SGX architecture includes a sealing capability, which allows data to be stored on persistent storage in an encrypted form. The sealing key is generated using a Key Derivation Function (KDF) [15] from the embedded base data seal key, which is fused into each processor during manufacturing by Intel. The sealing key is provided by the processor to the enclave, and only the enclave that sealed the data can later unseal it. SGX also supports remote attestation to allow a remote party to verify an enclave and establish a secure channel to it. To launch a production enclave in SGX version 1 (SGXv1), SGX required either the enclave binary or its author to be registered with Intel. In SGXv2, Intel added a flexible launch control capability, allowing the platform owner to bypass Intel as an intermediary in the enclave launch process [26, 51].

3 RANSOMWARE KEY MANAGEMENT

In this section, we discuss how ransomware key management schemes have evolved over the years. We then analyse the weaknesses of existing schemes in the face of modern anti-ransomware techniques, and propose five requirements of an attacker for a ransomware design. Notation used in our analysis and the remainder of the paper is as follows: (i) AK_{prv} (attacker's private key); (ii) AK_{pub} (attacker's public key); (iii) VK_{prv} (victim specific private key); (iv) VK_{pub} (victim specific public key); (v) $EK_{v(f)}/EK_v$ (symmetric key for victim v's file f — index f omitted where not relevant); (vi) *nonce* (pseudo-random number).

3.1 The Evolution of Ransomware Key Management

At the heart of any ransomware operation lies key management: if poorly implemented, it can potentially leak encryption keys to the victim and threaten the entire campaign [14]. Table 1 summarises our analysis of key management schemes in some of the most prevalent ransomware families. Our analysis extends several previous analyses [8, 23, 58], in particular, we also show existing core ransomware functionalities and match them against a set of requirements (formally defined in §3.2) that must be met by ransomware variants.

Our research shows that the majority (60%) of ransomware variants rely on an active Internet connection on the victim's machine to exchange keys with a command-and-control (C2) server [6, 8]. Additionally, 40% of ransomware variants (e.g., WannaCry) generate keys on a victim's untrusted machine, thus leaving them exposed and vulnerable to recovery [8, 23]. Older strains of ransomware,

Table 1: Ransomware key management requirements

Encryption Method	Families	Release	Ransomware RQ (§3.2)				
			RQ1	RQ2	RQ3	RQ4	RQ5
Symmetric	Apocalypse	2016	✗	✗	✗	✓	✗
	Jigsaw	2016	✗	✗	✗	✓	✗
	Razy	2016	✗	✗	✗	✓	✗
Asymmetric	CryptoLocker	2014	✓	✓	✗	✗	✗
	CBT-Locker	2014	✗	✗	✗	✓	✗
	CryptoDefense	2015	✗	✓	✗	✗	✗
	PetrWrap	2016	✓	✗	✗	✓	✗
	Unlock92	2016	✗	✗	✗	✓	✗
Hybrid	CryptoWall	2014	✓	✓	✗	✗	✗
	TorrentLocker	2014	✗	✗	✗	✓	✗
	TeslaCrypt	2015	✗	✗	✗	✓	✗
	Cerber	2016	✗	✓	✗	✓	✗
	Locky	2016	✓	✓	✗	✗	✗
	Petya	2016	✓	✓	✗	✗	✗
	RYUK	2017	✓	✗	✗	✗	✗
	WannaCry	2018	✗	✗	✗	✓	✗
	EKING	2020	✓	✗	✗	✓	✗
TEE	RansomClave	2021	✓	✓	✓*	✓	✓

* Only in proactive variant

such as Apocalypse and CryptoLocker, use either symmetric key or asymmetric key encryption. Newer strains, such as CryptoWall, TeslaCrypt, WannaCry and RYUK often opt for a hybrid encryption method. As Figure 1 shows, there are three variants, V1, V2, and V3 for key generation. In V1, ransomware, once executed (①), generates VK_{prv} and VK_{pub} locally on the victim's machine while sending the VK_{prv} to the attacker's C2 server. In V2, the ransomware generates the key pair on the C2 server, sending the VK_{pub} to the victim's

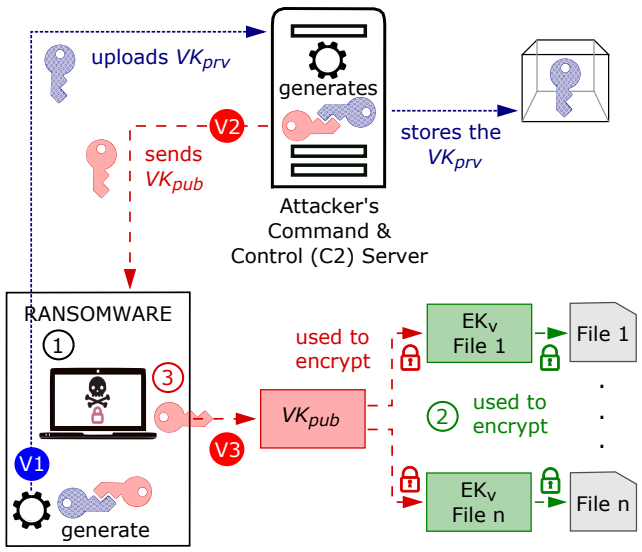

Figure 1: Current key management process of a typical ransomware

machine while safely storing the VK_{prv}. A unique EK_v is generated for each victim file to be encrypted (②), and all EK_v are then encrypted with the VK_{pub} (③).

3.2 Ransomware Key Management Requirements

We define a set of desirable requirements from an attacker's perspective that modern ransomware must fulfil to achieve its primary goal successfully, i.e. encrypt victim's data, secure the VK_{prv} and hold the data hostage until the ransom is paid. As we will see however, some of these requirements are potentially in conflict with each other.

RQ1: Secure asymmetric key generation. Ransomware must ensure that the VK_{prv} needed to decrypt individual EKv is not exposed to the victim at any point before the ransom is paid. Many attackers prefer setting up their own trusted C2 server to generate, distribute and securely store VK_{prv} as it provides some measure of protection against the arbitrary key recovery. However, this setup requires an outbound connection before beginning the encryption process [6, 18], which could be blocked/inspected by network monitoring tools [17, 42]. In addition, if the server's location is compromised, it could potentially be taken down by law enforcement [47], thus exposing VK_{prv} [40].

RQ2: Unique key per victim. Ransomware must generate a unique VK_{prv} for each victim to maximise the ransom payment received from victims. Some attackers deploy ransomware V3, such as IEncrypt, with an embedded VK_{pub} to eliminate outbound connections. However, using one global VK_{prv} could lead to victims collaborating and sharing the master key [45].

RQ3: Secure symmetric key generation. Ransomware in a hybrid encryption setting must ensure that the unique EK_v used to encrypt each victim file are not exposed to the victim at any point.

RQ4: No post-exploitation connections. To evade network monitoring software, Ransomware must avoid outbound connections to a C2 to retrieve the private keys or payload, or to an attestation service, until the victim's files have been encrypted.

RQ5: Trustworthy key release. Ransomware must employ a secure and reliable key release scheme post-attack to incentivise victims to co-operate and pay the requested ransom [12]. The scheme should also avoid unnecessary operational overhead for the attacker (e.g. to maintain an online presence, or manage large numbers of decryption keys). Some attackers request victims to send the encrypted VK_{prv} along with the ransom payment to avoid exposing the AK_{prv}. However, keys are still generated on the victim's machine leaving them vulnerable.

4 THREAT MODEL

In our threat model we assume the malware distribution and installation stages have been successfully performed. We assume the ransomware has the appropriate privilege to start enclaves and issue syscalls. We assume that the victim's machine supports Intel SGX and the SGX feature is enabled. We revisit the plausibility of these assumptions in our discussion of mitigations in §8.

We consider two different types of target systems: (i) standard system equipped with an AV, however, lacking any specific anti-ransomware functionality; (ii) advanced system equipped with an up-to-date AV that also includes an anti-ransomware module and can also capture cryptographic keys generated locally (e.g., PayBreak). In all cases, we assume that a victim cannot break the hardware-enforced security of SGX to obtain enclave-specific keys (e.g., sealing keys). We also assume that Intel would not release the key embedded into each processor during the manufacturing of the SGX-capable CPU [2].

5 RANSOMCLAVE KEY GENERATION AND ENCRYPTION

As a basis for security and performance analysis of enclave-enhanced ransomware, we present RansomClave, a family of ransomware that securely manage their cryptographic keys using an enclave. In this section we describe RansomClave's key generation and encryption phase, and how conflicts between the requirements introduced in §3.2 motivate two different RansomClave variants. We defer detailed discussion of trustworthy key release (RQ5) to §6.

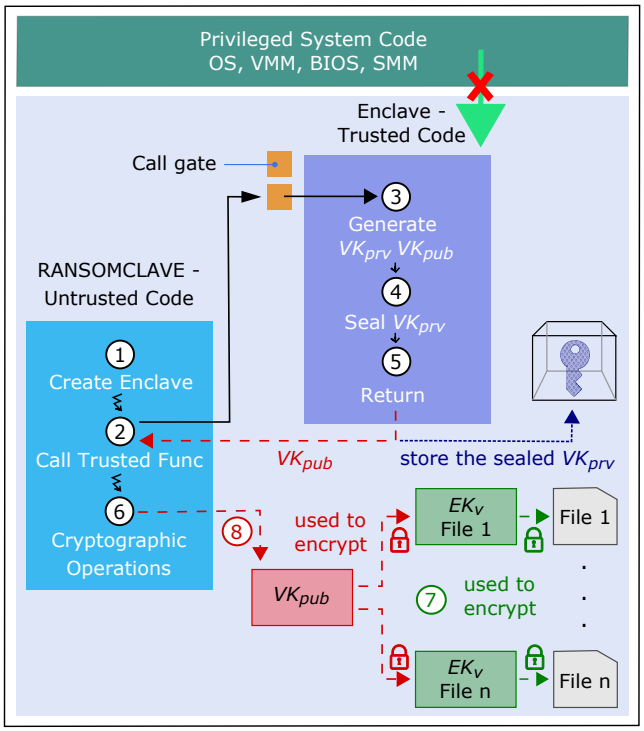

Figure 2: Key management and execution flow of RansomClave (steps ⑦-⑧ shown for reactive RansomClave)

RansomClave's key generation and encryption can be split into two stages. The first stage, asymmetric key generation (Figure 2, steps ①-⑤), is common to both RansomClave variants. The second stage, symmetric key generation and encryption, differs between variants (Figure 2, steps ⑦-⑧).

5.1 Asymmetric Key Generation

As shown in Figure 2, the RansomClave malware consists of both untrusted code (host application) and trusted code (enclave). RansomClave asymmetric key generation happens primarily inside the enclave. Once the malicious payload is executed, RansomClave's untrusted code creates the enclave ①. The untrusted code then performs an `ECALL` to enter the enclave and initiate key generation and encryption ②. On entering the enclave for the first time, RansomClave generates a victim specific asymmetric key pair (VK_{prv}, VK_{pub}) ③. RansomClave then creates an enclave-specific seal key using the `EGETKEY` instruction, and seals VK_{prv} using the seal key ④. Finally, the enclave performs an `OCALL` to send the unique VK_{pub} and sealed VK_{prv} to untrusted code ⑤. Both keys are then stored outside the enclave on the victim's system.

5.2 Symmetric Key Generation and Encryption

RansomClave's symmetric key generation and encryption stage is similar to other modern ransomware that use a hybrid scheme for improved performance. However, we introduce two different RansomClave variants for managing symmetric keys, reactive and proactive, each focused on the capabilities of the target system.

Reactive RansomClave: This variant generates symmetric keys and encrypts the victim's files *outside* the enclave. It targets a reactive victim who is unlikely to detect the ransomware and capture symmetric keys exposed in untrusted memory (RQ3) before encryption completes. However, encrypting files outside the enclave avoids additional data movement overheads, allowing encryption to complete more quickly and thus serving as a baseline for performance comparison with other variants.

Figure 2 shows the untrusted code generating a unique 256-bit AES symmetric key ($EK_{v(f)}$) for each file f before the encryption process starts (⑦). After encryption of file f completes, each $EK_{v(f)}$ is encrypted using VK_{pub} (⑧) and then the plaintext $EK_{v(f)}$ is deleted. Only the encrypted version of $EK_{v(f)}$ remains after the encryption process. Steps (⑦) and (⑧) repeat until all the victim files have been encrypted. The benefit of using per-file symmetric encryption keys is that the $EK_{v(f)}$ for files already encrypted will not be recoverable (since the original EK_v is deleted after encrypting each file) even if the ransomware attack is discovered and stopped mid-execution.

Proactive RansomClave: This variant assumes a proactive victim who is vigilant and potentially able to start capturing cryptographic keys soon after the ransomware is executed [19] It generates VK_{prv}, VK_{pub} and each $EK_{v(f)}$ inside the enclave. In addition, the ransomware also encrypts each victim file inside the enclave. By generating the EK_v and encrypting files inside an enclave, the proactive variant mitigates some of the security weaknesses of the reactive variant, i.e. exposure of the EK_v in memory (RQ3)

Although the proactive variant comes closer to meeting RQ3, we note that by default SGX does not provide confidentiality to enclave code. As a result, a proactive victim with sufficiently advanced AV could inspect the malware code and detect RansomClave given an appropriate signature. Although an additional RansomClave variant with private dynamic code loading is possible [53], it would need to communicate with a C2 server to download an encrypted

malicious payload (violating RQ4). For similar reasons, we do not consider using remote attestation to verify the enclave.

Another potential drawback of the proactive variant is its performance, since it must transfer victim files to the enclave from the host file system. Performing file related system calls from the enclave requires additional costly enclave transitions, potentially slowing the overall encryption process and jeopardising the attacker's primary goal of holding the victim's data hostage quickly. However, previous work has shown that asynchronous system calls can be used to mask much of this overhead [4]. We evaluate the overhead of the proactive variant with this optimization in §7.

6 RANSOMCLAVE KEY RELEASE

Once RansomClave has successfully encrypted the victim's files and presented the victim with a ransom note, the next phase of its operation is key release. As discussed in §3.1, existing key release schemes introduce additional security risks and operational overhead for the attacker. Furthermore, they require the victim to trust the attacker to release the decryption key after a ransom is paid.

In this section we show how RansomClave enables more robust key release schemes in comparison to existing ransomware. We next explore three alternative RansomClave key release schemes, each providing different trade-offs between the security of the scheme, the operational overhead of the attacker for managing decryption keys, and the likelihood the victim will pay the ransom[1].

6.1 Blockchain with Online Attacker

Our first RansomClave key release scheme leverages a public blockchain (e.g. Bitcoin) and digital signatures to exchange metadata such that RansomClave will only unseal VK_{prv} after the ransom has been paid.

Security. From the attacker's perspective, the advantage of this approach is its security, since the attacker can independently check a ransom payment exists on the blockchain. The disadvantages are that it requires the attacker to maintain an online presence to monitor the blockchain, increasing operational overhead, and it requires the victim to actually trust the attacker to release the key after a payment has been made (as with current ransomware). We note that, unlike the next two schemes proposed later in this section, a variant of this scheme could potentially be used in combination with existing (non SGX-based) hybrid encryption ransomware.

Overview. In this scheme, the victim is required to first make a ransom payment on the Blockchain that contains the following metadata in the transaction: AK_{pub}, VK_{pub}, and a *nonce*. Additionally, RansomClave derives a bitcoin wallet address based on the AK_{pub} embedded in it and generates a random *nonce* unique to the RansomClave enclave with high probability. The *nonce* is used as input to a cryptographic hash function digitally signed later on the attacker's machine once the attacker notices the ransom payment. (AK_{prv}, AK_{pub}) key pair is generated/stored on the attacker's machine while (VK_{pub}, VK_{prv}) is generated inside an enclave and stored (VK_{prv} as sealed) on the victim's machine. The victim next makes a ransom payment through Bitcoin [32]. Once the attacker notices a new payment is on the blockchain, he creates a new Bitcoin transaction providing some uniquely identifiable piece of metadata, such as the *nonce* signed using (AK_{prv}). RansomClave validates the authenticity and integrity of this data using the embedded AK_{pub}. If the verification is successful, the encrypted VK_{prv} is unsealed and released to the victim.

Key Release Steps. The blockchain and digital signature payment verification scheme is illustrated in Figure 3. We next describe the steps involved in detail:

- The victim follows the payment instructions (①) found on the ransom note (which also contains the bitcoin wallet address and the *nonce*) and deposits the requested amount of bitcoin into the newly generated wallet address (②). The victim also includes additional metadata consisting of the AK_{pub}, VK_{pub}, and *nonce* into the blockchain transaction using an `OP_RETURN` opcode (③)[2]. The *nonce* is generated and used along with VK_{pub} to give the attack a unique identification number. This is essential as it ensures another victim cannot use a previously obtained signed message hash from the attacker to trick the enclave into unsealing the AK_{prv}. `OP_RETURN` is an instruction in the Bitcoin scripting language that allows users to attach metadata to a transaction and save it on the blockchain. It is used to allow the victim and the attacker to exchange metadata required to instruct the enclave to unseal the VK_{prv}.
- The attacker periodically monitors the blockchain ledger for the AK_{pub} in transaction metadata using a payment verification program. This program searches for the `OP_RETURNS` opcode in an output's `scriptPubKey`[3] (④).
- When a transaction containing AK_{pub} is found and the payment is verified, the attacker employs a cryptography library to create a message hash using the SHA-256 hash of the metadata provided by the victim (⑤). The hash is then encrypted with AK_{prv} (⑥) using the standard RSA signature algorithm to obtain the signature (the RSA encrypted message hash). This process cryptographically binds the signed hash with the AK_{prv}, thus allowing RansomClave to verify the signature using the embedded AK_{pub}. A new transaction is uploaded on to the blockchain containing the signed hash and VK_{pub} (⑦).
- The victim periodically searches for new transactions containing the VK_{pub} (e.g. on blockchain explorer websites, such as blockchain.com) as per instructions provided in the ransom note. Once a transaction is found, the victim downloads the signed hash and submits it to RansomClave along with the original metadata consisting of AK_{pub}, VK_{pub} and *nonce* (⑧).
- RansomClave performs an `ECALL` (⑨) to enter the enclave which computes a hash of the original metadata. The algorithm then decrypts the message signature with the public key exponent of AK_{pub} to obtain the hash computed by the attacker. The two hashes are compared and if they match, RansomClave considers the signature valid. RansomClave then unseals and releases VK_{prv} to the victim.

[1] In all these schemes, for simplicity, we assume that, once the private key has been released, the victim is able to decrypt the files autonomously, e.g. by using a decryptor that takes as input the private key and that is part of RansomClave or provided by the attacker independently.

[2] We assume the victim is provided with detailed instructions to do so or uses an automated tool to perform the necessary payment steps autonomously and easily.

[3] A locking mechanism placed on an output to prevent others from spending the bitcoin.

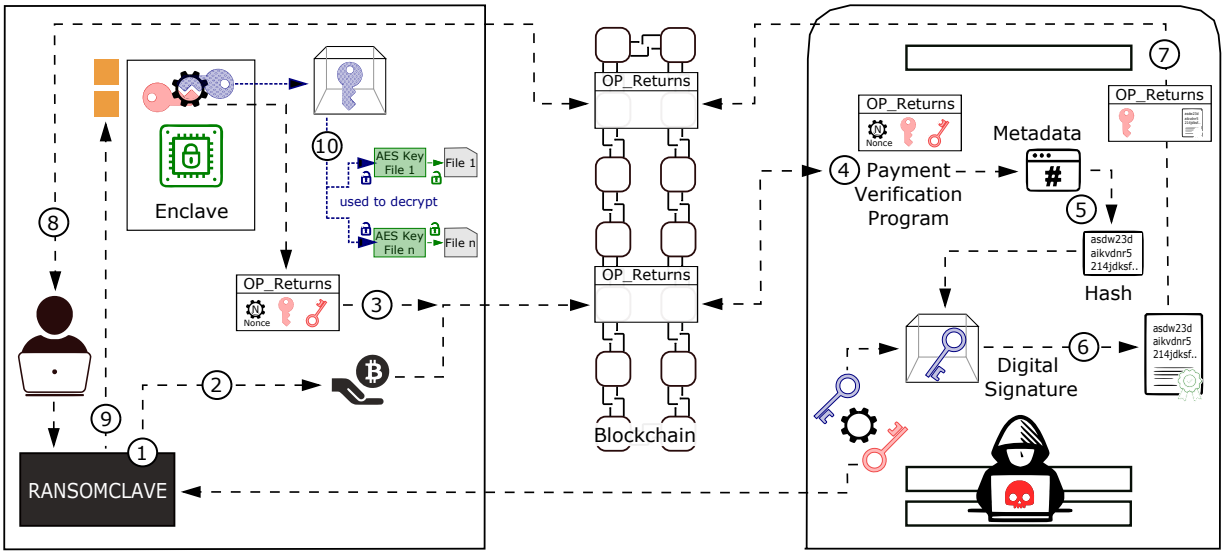

Figure 3: Key Release Scheme #1: Blockchain with Online Attacker Payment Verification Diagram

6.2 Enclave SPV Client

To eliminate the attacker's online presence, we next propose a fully *autonomous* and *offline attacker* (*trustless*) solution that uses a Simplified Payment Verification (SPV) client inside the enclave to verify ransom payments on the blockchain. In this scheme, RansomClave is deployed with a lightweight blockchain client embedded in it. This allows RansomClave to successfully verify the victim's payment on the blockchain and release VK_{prv} without inputs from the attacker.

Security. The main advantage of this scheme is that it is fully autonomous. This reduces operational overhead for the attacker, who no longer needs to maintain an online presence. More importantly, full autonomy means the victim only needs to trust[4] the enclave code, whose behaviour post-payment can be verified up front using attestation. This is a very attractive property for the attacker potentially, since it can increase the chances of infected victims paying the ransom. On the downside, this scheme has arguably weaker security. In fact, enclave interactions with a blockchain are potentially vulnerable to man-in-the-middle (MitM) attacks [34]. Since the victim has full control over the enclave's network connectivity, it can impersonate the blockchain to trick the enclave into accepting fake blocks. However, mining plausible fake blocks still incurs an economic cost for the victim[5]. As we will show, in many cases it is practical for the attacker to increase this cost such that it is cheaper for the victim to pay the ransom.

Overview. Currently, there are two primary methods of validating the blockchain as a client: full nodes and SPV clients. A full node stores a copy of the entire blockchain with all the transactions, from the genesis block to the most recently discovered block. In contrast, SPV clients can check if particular transactions are included in a block through block headers only (80 bytes per block) and a Merkle tree rather than downloading the entire block [3, 11].

Our second RansomClave key release scheme embeds an SPV Client inside the enclave, together with an embedded hash of the most recent stable block at the time the attacker creates the malware. The SPV client inside the enclave then requests full nodes to supply it with the headers of all blocks since the block corresponding to the embedded hash, in addition to a Merkle path for the transaction containing the ransom payment. Once the SPV client is satisfied the payment has been made, the enclave releases VK_{prv} to the victim. Although full nodes usually give stronger security than SPV clients, for RansomClave this is not the case. Since the victim has full control over the attacker's network connectivity, it can mine a new block on top of the current longest Bitcoin chain with a fake transaction containing a supposed ransom payment. The victim must simply execute the standard Bitcoin proof-of-work mining algorithm to find a suitable nonce value. Even if the malware contains a full node, the hash of the genesis block, and the most recent block at the time the malware is created, the enclave cannot distinguish subsequent blocks mined by the victim from blocks on the real Bitcoin blockchain. RansomClave therefore avoids the additional storage and synchronisation overhead of a full node using an SPV client.

Depending on the size of the ransom, it may be cheaper for the victim to mine a fake block than pay the ransom. As a countermeasure, the attacker can require the victim to provide a valid chain with n additional blocks mined on top of the block containing the ransom payment transaction. The victim would then need to mine a fake *chain* of length n, linearly increasing the cost to fake a ransom payment. However, this mitigation would also require "honest" victims to wait for n blocks after their payment has been made, and thus may not be practical for very large ransoms if timely key release is important. In this strategy, the attacker's bitcoin address

[4]We define trust as a utility function where we express implicit trust by paying would be an expectation that the victim receives something in return, i.e. the decryption key.
[5]The estimated cost of mining a Bitcoin block in 2018 was between $2k-$20k depending on the cost of electricity, e.g. see: https://www.marketwatch.com/story/heres-how-much-it-costs-to-mine-a-single-bitcoin-in-your-country-2018-03-06.

is embedded in the RansomClave binary. Additionally, the victim must include the unique nonce generated by the enclave into the bitcoin transaction. RansomClave will use this metadata to verify the uniqueness of the transaction prior to unsealing the AK_{prv}.

Key Release Steps. As shown in Figure 4, RansomClave executes the SPV client inside an enclave which in turn connects to a full Bitcoin node. We next describe the steps in detail: (i) after RansomClave receives the payment transaction from the victim (①), it loads the sealed victim's wallet address into enclave and establishes a TCP connection (②) to the full Bitcoin node. (ii) the SPV client sends a *getblock* message (③) to request a list of blockchain headers from the current block number and hash embedded in RansomClave when the attacker created it. (iii) the full node checks transactions and sends a *merkleblock* message (④), which comprises of block headers of the blocks and the Merkle path of the matched transaction. (iv) the SPV client (⑤) uses the *merkleblock* to verify that the ransom payment transaction is included in the blockchain and has more than six confirmations. If the verification is successful, then the VK_{prv} is unsealed and released.

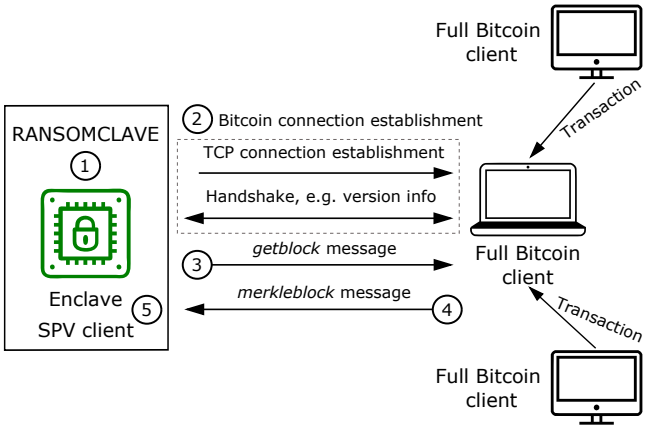

Figure 4: Key Release Scheme #2: Enclave SPV Payment Client Verification Diagram (a revised version from [20])

6.3 Blockchain Explorer Client

To further improve the security of the SPV client scheme, we propose an alternative autonomous key release scheme that uses a third-party *blockchain explorer service*[6]. Blockchain explorer services allow clients to retrieve transaction(s) relating to a particular bitcoin address over a secure HTTPS channel.

Security. This approach provides security against straightforward MitM attacks. To subvert it, the victim must convince the blockchain explorer service to provide fake data to the enclave (or pressure law enforcement to force them). Even if this is possible, the attacker can combine this scheme with the previous SPV client scheme such that the victim must still pay the cost required to mine plausible fake blocks. One downside of this approach is that it introduces an additional dependency on a third party service. If this becomes unavailable for any reason post-infection, even an "honest" victim will no longer be able to retrieve the decryption key VK_{prv} from the enclave.

Key Release Steps. In this scheme, when RansomClave is executed, the enclave initiates a TLS connection with the Blockchain explorer API server. We describe the steps in detail: (i) Once the TLS Handshake is complete, the server's digital certificate is retrieved along with information about the Certificate Authority (CA) that issued it. The digital certificate contains the public key and the identity of the owner. Additionally, it is signed by the CA using its private master key. This allows the enclave to use a pre-seeded store of SSL certificates authorities' public keys embedded in the binary to verify integrity and authenticity of the retrieved certificate. (ii) Once a secure channel has been established, the enclave makes an API call to the Blockchain explorer API server for transactions and block data relating to the attacker's Bitcoin addresses. (iii) The server sends relevant data to the enclave. If a valid transaction with a minimum of six confirmations is found, then the enclave releases the VK_{prv}. Six confirmations is generally accepted for most transactions as it represents enough security to ensure the validity of the transaction.

7 IMPLEMENTATION AND EVALUATION

We present a proof-of-concept implementation of RansomClave design. Our prototype implementation operates similarly to WannaCry but executes inside an Intel SGX enclave. The reactive variant of RansomClave is composed of 2,600 SLOC, while for the proactive variant, the SGX-LKL library OS is also needed as well as code restructuring for symmetric key generation.

Algorithm 1: Setup and Execution of RansomClave

```
Result: Secret launchToken for the enclave, createEnclave
        and encryptFile
Input: enclaveId;
launchToken ← secretKey();
if createEnclave(launchToken, enclaveId) then
    VK_prv, VK_pub ← enclaveGenKey();
    VK_prv ← seal(enclaveId, VK_prv);
    VK_pub ← save(enclaveId, VK_pub);
else
    abort();
VK_pub ← importKey();
for file f in getImportantFiles() do
    EK_v(f) ← generateSymKey() ;
    encrF ← encrypt(f, EK_v(f));
    encrSymKey ← encrypt(EK_v(f), VK_pub);
    fConcat ← append(encrSymKey, encrF);
    delete(f);
    save(fConcat);
displayRansomNote();
```

Experimental Setup.. All our experiments were performed on Intel® Dual-Core™ i5-7360U CPU with 4GB of RAM, and 3 CPU cores at 2.30GHz. Our system ran Ubuntu 18.04.5 LTS 64-bit within

[6]For example, Blockchain.com provides an explorer API at https://www.blockchain.com/api.

VirtualBox 6.0. In addition, we performed our tests using Intel SGX drivers [25], the Intel SGX SDK [24], and the open-source SGX-LKL library OS [35].

RansomClave. Algorithm 1 describes the RansomClave program in reactive variant. In detail, when the malware is executed, its first task is to initiate the enclave using a launch token and get the buffer sizes required by the untrusted code to store the VK_{pub} and sealed VK_{prv}. Once the memory is allocated, the untrusted code makes an `ECALL` by calling the `enclaveGenKey` function to generate (VK_{prv}, VK_{pub}) pair using 256-bit Elliptic Curve Cryptography (ECC). The enclave seals VK_{prv} using the data seal key and returns both keys to the untrusted code. When the keys are in place, RansomClave initiates the encryption process by calling the `getImportanFiles` function, which starts searching for the victim's files in specified targeted folders and encrypting each file f using a new $EK_{v(f)}$. Each $EK_{v(f)}$ is then encrypted using VK_{pub} before it is permanently deleted. The encryption process is repeated until there are no new files to encrypt. Finally, a ransom note is displayed to the victim containing instructions on how to make the payment and unseal VK_{prv} to restore the files.

Algorithm 2: Enclave Launch and Key Release of RansomClave

```
Result: Launch enclave, initiate TLS connection, get
        transaction data, verify and unseal VK_prv
Input: enclaveId, hostName, portNum, walletAddr, txId;
launchToken ← secretKey();
if loadEnclave(launchToken, enclaveId) then
    SSL_Library_init();
    ctx ← initCTX();
    server ← openConnection(hostName, portNum);
    ssl ← sslNew(ctx);
    sslSetFd(ssl, server);
    if (sslConnect(ssl) == FAIL) then
        abort();
    else
        txResult ← getTx(hostName, walletAddr, txId);
        if (verify(txId) == TRUE) then
            VK_prv ← unseal(enclaveId, VK_prv);
            release(VK_prv);
        else
            print("Verification failed");
```

Proactive RansomClave. We extend our proof-of-concept RansomClave to support the proactive variant using the open-source SGX-LKL library OS [35], which is designed to run existing unmodified Linux binaries inside enclaves. SGX-LKL executes the ransomware sample to perform cryptographic operations on sample victim data (provided as part of a Linux disk image) inside an enclave. The in-enclave SGX-LKL library OS provides transparent system calls (e.g. file and network I/O), user-level threading, and signal handling. Reactive RansomClave is categorised as native since its cryptographic operations are performed in user-space similar to traditional ransomware operation.

Blockchain Explorer Client. Algorithm 2 describes the blockchain explorer services with offline attacker scheme we have implemented within RansomClave. When RansomClave is executed, the victim is asked to provide a bitcoin address along with the transaction identifier (`txId`) of the ransom payment. An enclave is then loaded using a launch token followed by an `OCALL` to initiate a TLS handshake with the Blockchain API server. The `getTx` message is sent to the server to request a data block linked to `txId` and wallet address (`walletAddr`) along with number of confirmations. If a valid transaction with at least six confirmations is received, then the enclave executes the `unseal` function to unseal and release VK_{prv}.

Operational and Performance Evaluation. We have tested the proof-of-concept RansomClave operational capabilities on four different SGX-enabled VMs, each configured to store different types of files as to mimic user's and organizational scenarios. In each run, RansomClave encrypted 200 files whose size ranged from few KB up to 250MB. After the encrypting operations, the user was shown a ransom note requesting a symbolic sum, and we performed the necessary steps to perform key release.

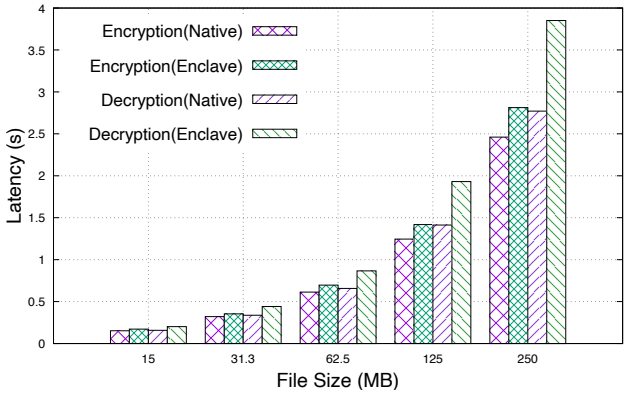

Figure 5: Cryptographic operation performance evaluation

Our performance evaluation compares the speed of cryptographic operations for the two RansomClave variants proposed in this paper, reactive (native) and proactive (enclave). Figure 5 shows the measurement of cryptographic operations on files whose size ranges from 15MB to 250MB: we can see that these operations in the enclave take longer to complete when compared to the native ransomware. An average of 12.76% overhead for the encryption operation and 34.05% in decryption operation is observed when the ransomware sample is executed inside an enclave. Additionally, the results also revealed that the decryption operation is slower for both the native and enclave ransomware. This could be potentially due to the implementation of AES in Cipher Block Chaining mode, which uses cipher in the inverse direction, thus resulting in slower decryption timings [48]. However, it is possible to achieve faster decryption on a system that supports parallel decryption, although this may result in using more memory due to loading multiple copies of the encrypted block.

Key Capture. We tested RansomClave's key management against popular memory forensics tools and techniques, such as Dumpit, Volatility, and PayBreak [31]. Our experiments confirm it is possible

to extract keys from memory for a reactive variant that generates symmetric keys outside of an enclave. However, it is not feasible for the proactive variant as all keys are generated inside an enclave.

8 DISCUSSION

RansomClave Pre-Infection Mitigations. Several methods can be applied to reduce RansomClave's attack surface. In particular, given that the main focus of RansomClave is to provide robust key management, some existing mitigations against ransomware based on static analysis or in-memory analysis techniques [5, 8, 17, 29, 31, 42, 43, 55] can detect RansomClave. For instance, binary inspection of an unknown program could detect cyptographic calls [22] or file calls [56]. In addition, as shown in [7, 50], it is also possible to detect unknown malicious code and functions in enclave binaries using existing static analysis techniques, since SGX by default does not provide code confidentiality. Furthermore, a malicious program can only issue system calls through the host application: as such, software monitoring system calls could stop the malware before it can take any destructive actions. A user could also disable SGX functionality in the system BIOS when not in use, although this is undesirable. Lastly, system privilege is needed to create an enclave. As such a strict policy could be enforced on the end user machine to stop RansomClave execution.

Launch Control. Intel SGX's launch control whitelist provides a mechanism to prevent wide-scale abuse of enclaves on systems where it is straightforward to gain admin privileges. However it requires Intel to proactively monitor and revoke whitelisted keys that have been compromised or abused, and for systems to regularly update their local copy of the whitelist. We note however that openness concerns around launch control have hindered upstream support for SGX in the Linux kernel. The recent availability of Flexible Launch Control on newer versions of SGX mitigates some of these openness concerns, but shifts the burden for securely managing the whitelist to the system owner, and potentially requires a secure-boot mechanism to protect the whitelist against an attacker with admin privileges. To the best of our knowledge, other enclave technologies (e.g. AMD SEV) do not have a hardware vendor-enforced launch control whitelist.

RansomClave Post-Infection Mitigations. As shown by [13, 44], SGX is vulnerable to software-based side-channel attacks that, according to Intel, can only be prevented by the developers themselves[7], i.e., Intel does not consider side channel attacks as part of the SGX threat model. Hence, victims infected with RansomClave could potentially extract ransomware's keys from inside of an enclave via side-channel attacks.

Extensions. We have considered RansomClave targeting multiple, distinct users. In an enterprise scenario, rather than creating enclaves on multiple hosts, RansomClave can set up a single internal SGX server to manage all the cryptographic keys of the infected hosts on a local network. In this instance, a clear advantage from an attacker's perspective is that it simplifies the key management, particularly if some hosts do not have SGX support. However, relying on a single machine to generate all keys could impact untrusted code's ability to promptly encrypt victim's files since the encryption key is not immediately available. Additionally, it also leads to a single point of failure for the attack.

Ethical Considerations. This research has ethical concerns of dual use research as our findings could improve the key management of modern ransomware. However, by commenting on the theoretical risks of future SGX-based ransomware, we preemptively foster discussions on the possible risks of SGX-empowered malware that might have been underestimated [51]. We also omit implementation details to develop a fully-fledged SGX-based ransomware and key release scheme – this would require cyber-criminals to perform a non-trivial integration task – and outline mitigation improvements.

9 RELATED WORK

SGX Malware SGX-ROP is the first attack to enable an enclave malware to stealthily control its host application independent of the enclave API [51]. SGX-ROP uses an Intel Transactional Synchronisation Extensions-based technique to construct a write-anything-anywhere primitive and a memory-disclosure primitive from inside an enclave. SGX-ROP can bypass ASLR, stack canaries, and address sanitiser, to run ROP gadgets in the host context, enabling practical enclave malware. However it does not evaluate the implications for key release. Marshalek discusses the possibility of ransomware running inside enclaves, but does not evaluate performance overhead or the implications for key release[38].

Enclave Interaction with Blockchain. Authors [21] proposes using blockchain to implement one-time programs via cryptographic obfuscation techniques. [27] expands on this idea by proposing a protocol that uses an enclave to facilitate secure state management for randomised multi-step computations. It also introduces the concept of combining the enclave with public ledgers to condition a program execution on the publication of particular messages on the ledger, and describes a practical set of applications leveraging this protocol. In contrast, our research provides an in-depth analysis of the trade-off and security advantage between different blockchain-enabled key release schemes.

SGX-based Key Management Frameworks. Some research efforts [10, 46] explore the application of SGX in a cloud environment to minimise the disclosure of sensitive data to a third-party hosting and providing the cloud services. Authors [46] proposes key-management frameworks for data-centric networking where an SGX-based key server, which supports remote attestation and key sealing, generates and manages data encryption and decryption keys. Neither work addresses the implications of SGX for ransomware key management.

10 CONCLUSION

Based on an analysis of key management schemes of existing ransomware, we observe that most ransomware strains either generate keys on the victim's machine, thus leaving them vulnerable to memory key extraction techniques, or require contact with a C2 server. However, recent work has raised the spectre of future enclave-enhanced malware that can avoid such mitigations using emerging

[7] E.g., see: https://software.intel.com/content/www/us/en/develop/articles/intel-sgx-and-side-channels.html

support for hardware-enforced secure enclaves. Given the damage ransomware causes, it is important to understand what new advantages such ransomware might provide to attackers.

We implement a proof-of-concept called RansomClave to demonstrate the practicality of such ransomware emerging in the near future. Our evaluation shows that RansomClave can provide a practical key management solution with minimal overhead to address known weaknesses in existing ransomware strains. RansomClave also raises the possibility of autonomous blockchain-based key release schemes that no longer require victims to trust the attacker to release decryption keys or the attacker to maintain an online presence, potentially increasing attacker profitability. We show however that a variety of mitigation techniques are still possible for a security-aware victim.

ACKNOWLEDGMENTS

This research of Alpesh Bhudia is supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/P009301/1).

REFERENCES

[1] B. Al-rimy et al. 2018. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Comput & Sec.* (2018).
[2] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *HASP '13.*
[3] Andreas M Antonopoulos. 2014. *Mastering Bitcoin: unlocking digital cryptocurrencies.* O'Reilly Media, Inc.
[4] Sergei Arnautov, Bohdan Trach, Franz Gregor, et al. 2016. SCONE: Secure Linux Containers with Intel SGX. In *OSDI '16.*
[5] P. Bajpai and R. Enbody. 2020. Memory forensics against ransomware. In *Int. Conference on Cyber Security '20.*
[6] P. Bajpai, A. K. Sood, and R. Enbody. 2018. A key-management-based taxonomy for ransomware. In *APWG eCrime '18.*
[7] J. Bergeron, Mourad Debbabi, J. Desharnais, et al. 2009. Static Detection of Malicious Code in Executable Programs. *Int. J. of Req. Eng* (2009).
[8] Eduardo Berrueta, Daniel Morato, Eduardo Magaña, et al. 2019. A Survey on Detection Techniques for Cryptographic Ransomware. *IEEE Access* 7 (2019).
[9] Akashdeep Bhardwaj. 2017. Ransomware: A rising threat of new age digital extortion. In *Online Banking Security Measures and Data Protection.* IGI Global.
[10] Stefan Brenner, Tobias Hundt, Giovanni Mazzeo, et al. 2017. Secure Cloud Micro Services Using Intel SGX. In *DAIS '17.*
[11] B. Bünz, L. Kiffer, Loi Luu, et al. 2020. FlyClient: Super-Light Clients for Cryptocurrencies. In *IEEE Symposium on Security and Privacy (SP) '20.*
[12] E. Cartwright, J. Hernandez-Castro, and A. Cartwright. 2019. To pay or not: game theoretic models of ransomware. *J. of Cybersecurity.* (2019).
[13] Guoxing Chen, Sanchuan Chen, Yuan Xiao, et al. 2019. Sgxpectre: Stealing Intel secrets from SGX enclaves via speculative execution. In *IEEE EuroS&P '19.*
[14] M. Conti, A. Gangwal, and S. Ruj. 2018. On the economic significance of ransomware campaigns: A Bitcoin transactions perspective. *Comput. & Sec.* (2018).
[15] V. Costan et al. 2017. Secure processors part II: Intel SGX security analysis and MIT sanctum architecture. *Foundations and Trends in Electronic Design Automation* (2017).
[16] V. Costan and S. Devadas. 2016. Intel SGX Explained. *Cryptol. ePrint Arch.* (2016).
[17] Greg Cusack, Oliver Michel, and Eric Keller. 2018. Machine learning-based detection of ransomware using SDN. In *Int. Workshop on SDN-NFV Security '18.*
[18] Tooska Dargahi et al. 2019. A Cyber Kill-Chain based taxonomy of crypto-ransomware features. *J. Computer Virology and Hacking Techniques* (2019).
[19] Simon Davies et al. 2020. Evaluation of Live Forensic Techniques in Ransomware Attack Mitigation. *Forensic Science International: Digital Investigation.* 33 (2020).
[20] Arthur Gervais, Srdjan Capkun, et al. 2014. On the privacy provisions of bloom filters in lightweight bitcoin clients. In *ACSAC '14.* 326–335.
[21] Rishab Goyal and Vipul Goyal. 2017. Overcoming cryptographic impossibility results using blockchains. In *Theory of Cryptography Conference '17.*
[22] Felix Gröbert, Carsten Willems, et al. 2011. Automated identification of cryptographic primitives in binary programs. In *Int. Workshop on RAID '11.*
[23] J Alex Halderman, Seth D Schoen, Nadia Heninger, et al. 2009. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM* 52, 5 (2009).
[24] Intel. 2020. *Intel SGX for Linux.* Retrieved 2020-09-04 from https://github.com/intel/linux-sgx
[25] Intel. 2020. *Intel SGX Linux Driver.* Retrieved 2020-12-15 from https://github.com/intel/linux-sgx-driver
[26] Intel. 2020. Intel Software Guard Extensions (Intel SGX) Data Center Attestation Primitives: ECDSA Quote Library API. (2020).
[27] Gabriel Kaptchuk, Matthew Green, and Ian Miers. 2019. Giving State to the Stateless: Augmenting Trustworthy Computation with Ledgers. In *NDSS '19.*
[28] Masoudeh Keshavarzi and Hamid Reza Ghaffary. 2020. I2CE3: A dedicated and separated attack chain for ransomware offenses as the most infamous cyber extortion. *Computer Science Review* (2020).
[29] Amin Kharaz, Sajjad Arshad, Collin Mulliner, et al. 2016. UNVEIL: A large-scale, automated approach to detecting ransomware. In *USENIX Security '16.*
[30] SH Kok, Azween Abdullah, NZ Jhanjhi, and Mahadevan Supramaniam. 2019. Ransomware, Threat and Detection Techniques: A Review. *Int. J. CSNS* (2019).
[31] Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. 2017. Paybreak: Defense against cryptographic ransomware. In *ASIA CCS '17.*
[32] Xiaoqi Li, Peng Jiang, Ting Chen, et al. 2017. A Survey on the Security of Blockchain Systems. *Future Generation Computer Systems* (2017).
[33] Allan Liska and Timothy Gallo. 2016. *Ransomware: Defending against digital extortion.* O'Reilly Media, Inc.
[34] A. Loe and E. Quaglia. 2019. You Shall Not Join: A Measurement Study of Cryptocurrency Peer-to-Peer Bootstrapping Techniques. In *SIGSAG '19.*
[35] LSDS Group. 2020. *SGX-LKL Library.* Retrieved 2021-02-18 from https://github.com/lsds/sgx-lkl
[36] Robert Luo and Qinyu Liao. 2007. Awareness Education as the Key to Ransomware Prevention. *Information Systems Security* (2007).
[37] Carsten Maartmann-Moe et al. 2009. The persistence of memory: Forensic identification and extraction of cryptographic keys. *Digital Investigation* (2009).
[38] Marion Marschalek. 2018. The Wolf In SGX Clothing. *Bluehat IL* (2018).
[39] Frank McKeen, Ilya Alexandrovich, et al. 2013. Innovative Instructions and Software Model for Isolated Execution. In *Int. Workshop on HASP '13.*
[40] Nailah Mims. 2017. The Botnet Problem. In *Comput. Info. Sec. Handbook.*
[41] Saeid Mofrad, Fengwei Zhang, Shiyong Lu, and Weidong Shi. 2018. A comparison study of Intel SGX and AMD memory encryption technology. In *HASP '18.*
[42] Daniel Morato, Eduardo Berrueta, Eduardo Magaña, et al. 2018. Ransomware early detection by the analysis of file sharing traffic. *J. Netw. Comput. App.* (2018).
[43] Routa Moussaileb, Benjamin Bouget, Aurélien Palisse, et al. 2018. Ransomware's early mitigation mechanisms. In *DIMVA '18.*
[44] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. 2020. A survey of published attacks on Intel SGX. *ArXiv:2006.13598* (2020).
[45] H. Orman. 2016. Evil Offspring - Ransomware and Crypto Technology. *IEEE Internet Computing* (2016).
[46] M. Park, J. Kim, Y. Kim, et al. 2019. An SGX-Based Key Management Framework for Data Centric Networking. In *Int. WISA '19.*
[47] No More Ransom. 2021. *No More Ransom.* Retrieved 2021-03-14 from https://www.nomoreransom.org
[48] Phillip Rogaway. 2011. Evaluation of some blockcipher modes of operation. *CRYPTREC for the Government of Japan* (2011).
[49] M. Sabt, M. Achemlal, and A. Bouabdallah. 2015. Trusted Execution Environment: What It is, and What It is Not. In *2015 IEEE Trustcom/BigDataSE/ISPA.*
[50] Michael Schwarz, Samuel Weiser, et al. 2020. Malware Guard Extension: abusing Intel SGX to conceal cache attacks. *Cybersecurity* (2020).
[51] Michael Schwarz, Samuel Weiser, and Daniel Gruss. 2019. Practical enclave malware with Intel SGX. In *DIMVA '19.*
[52] Carlton Shepherd et al. 2017. Establishing Mutually Trusted Channels for Remote Sensing Devices with Trusted Execution Environments. In *ARES '17.*
[53] R. Silva, P. Barbosa, and A. Brito. 2017. DynSGX: A Privacy Preserving Toolset for Dinamically Loading Functions into Intel SGX Enclaves. In *Cloudcom '17.*
[54] Rohit Sinha, Sriram Rajamani, Sanjit Seshia, and Kapil Vaswani. 2015. Moat: Verifying confidentiality of enclave programs. In *ACM SIGSAC '15.*
[55] Sayan Sinha, Manaar Alam, Sarani Bhattacharya, et al. 2018. RAPPER: Ransomware Prevention via Performance Counters.
[56] K. P. Subedi et al. 2018. Forensic Analysis of Ransomware Families Using Static and Dynamic Analysis. In *IEEE Security and Privacy Workshops (SPW) '18.*
[57] Luis Javier García Villalba, Orozco, et al. 2018. Ransomware Automatic Data Acquisition Tool. *IEEE Access* (2018).
[58] James Wyke and Anand Ajjan. 2015. The current state of ransomware. *SOPHOS. A SophosLabs Technical Paper* (2015).